Int. J. on Recent Trends in Engineering and Technology, Vol. 9, No. 1, July 2013

Simulating Turing Machines Using Colored Petri Nets with Priority Transitions

A. Javan, M. Akhavan, and A. Moeini
Department of Algorithms and Computation, Faculty of Engineering Science, School of Engineering, University of Tehran, Iran
Email: {ajavan, moeini}@ut.ac.ir, m.akhavan@alumni.ut.ac.ir

***Abstract*—In this paper, we present a new way to simulate Turing machines using a specific form of Petri nets such that the resulting nets are capable of thoroughly describing behavior of the input Turing machines. We model every element of a Turing machine's tuple** (i.e., $Q, \Gamma, b, \Sigma, \delta, q0, F$) **with an equivalent translation in Colored Petri net's set of elements with priority transitions such that the resulting translation (is a Petri net that) accepts the same language as the original Turing machine. In the second part of the paper we analyze time complexity of Turing machine's input in the resulting Petri net and show that it is a polynomial coefficient of time complexity in the Turing machine.**

***Index Terms*— Turing machine, Colored Petri net, simulate, priority transition, time Complexity**

I. Introduction

Petri net is one of the several powerful yet simple mathematical models widely used for describing distributed systems . The application of Petri nets has been studied in variety of domains such as scheduling systems , process modeling , digital logic modeling , etc . A Petri net (PN) is structurally similar to a graph in which vertices and edges are replaced by places and transitions , respectively . On the other hand , Turing machine (TM) [3] is known as an extremely powerful tool for accepting languages . One well-known use of a Turing machine is to visualize an abstract Oracle machine in order to study/solve different decision problems (e.g . finding the set of prime numbers) or computational problems (e.g . cryptography) . Another well-used form of the machines is non-deterministic Turing Machine (NDTM) which have the capability of solving many different problems (e.g . decidability) . Even though Turing machines are powerful and able to compute any function in a discrete mathematical form (implied by the Church-Kleene's thesis[4][5]) , but there are cases (e.g . NP-complete problems in graph domain such as Hamiltonian cycle or min-cut) where these machines are somehow hard to be used and difficult to be understood .

The extended Petri net or Petri net with inhibitor arcs is one of the earliest extensions of the standard Petri net theory , first introduced in [1] in 1973 . Inhibitor nets (or nets with inhibitor arcs) [1] further generalize contextual nets with the possibility of checking not only for the presence , but also for the absence of tokens in a place . Agerwala et al . [2] in 1974 showed that Petri net with inhibitor arcs can simulate a Turing machine . In this paper , we also show that Colored Petri nets with priority transitions can simulate a Turing machine . Although with a little exploration into this matter , you can understand that Colored Petri net with priority transitions has the capabilities of Petri net with inhibitor arcs . In other words , we propose Petri machine (PM) , a specific type of Petri net which is a combination of simplicity of Petri nets and power of Turing machines . We present Petri machine to prove that Petri nets are as powerful as Turing machines for certain problems . We prove that , in the Colored Petri net [8] which is the type of Petri net used for this work , if transitions were prioritized , then it would be possible to transform a Turing machine into a Petri net . We refer to this translated version of Petri net as a Petri machine . Using the proposed transformation , we show that all the Turing machine's features are preserved and , for every one of features in a given Turing machine , there is a correspondence in the resulting Petri machine . After we show that Colored Petri nets with priority transitions has the capabilities of Turing machines , we analyze the time complexity of executing an instruction in both Turing machine and Petri machine .

The rest of this paper is organized as follows . In Section II , A brief overview and background knowledge is presented for Petri nets and Turing machines . Then , in Section III we describe in details how the Turing machine can be simulated by the Colored Petri net with priority transitions . In section IV , analysis of time complexity for executing the instruction in Turing machine and Petri machine is presented . Finally , the conclusion is drawn.

II. Background

In this section , we briefly overview the two tools on which this paper's primary contribution is based : Turing machines and Colored Petri nets .

A. Turing Machines

Turing Machine [3] is an automaton whose temporary storage is a single, one dimensional array of cells, each of which can hold a single symbol. This array is capable of holding an unlimited amount of information. The storage device of Turing Machine is called a tape because it is analogous to the magnetic tapes used in actual computers. Associated with the tape is a read-write head that can

DOI: 01.IJRTET.9.1.34

move right or left on the tape and that can read and write a single symbol on each move. A Turing Machine M is defined precisely by

$$M = (Q, \Gamma, b, \Sigma, \delta, q0, F)$$

Where Q is the set of internal states, Γ is a finite set of symbols called the tape alphabet, $b \in \Gamma$ is a special symbol called the blank, " is the input alphabet, δ is the transition function, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states.The transition function δ is defined as

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

In general, δ is a partial function on $Q \times \Gamma$. L . L and R stands for Left and Right when a new tape symbol is operated and head is replaced.

B. Petri Nets

Petri Net [**Error! Reference source not found.**6][7] is a graphical and mathematical tool with places, transitions and arcs. Places can hold token(s) which themselves might be colorful (i.e., in a colored Petri Net). Each arc is drawn from a place to a transition and from a transition to a place. Each transition gets "fired" when the sources of all the input arcs have sufficient number of tokens. A colored Petri Net [8] is a tuple CPN=(Σ,P,T,A,N,C,G,E,I) that satisfying the following requirements. Σ is a finite set of non-empty types, called color sets, P is a finite set of places, T is a finite set of transitions, A is a finite set of arcs such that $(P \cap T = P \cap A = T \cap A = \emptyset)$, N is a node function. It is defined from A into P×T*"T×P, C is a color function. It is defined from P into Σ , G is a guard function. It is defined from T into expressions such that

$$\forall t \in T : [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \sum]$$

E is an arc expression function. it is defined from A into expressions such that

$$\forall a \in A : \begin{bmatrix} Type(E(p)) = C(p(a))MS \\ \wedge \\ Type(Var(E(a))) \subseteq \Sigma \end{bmatrix}$$

Where p(a) is the place of N(a), $C(p(a))_{MS}$ denotes the set of all multi sets over C(p(a)), Type(expr) denotes the type of an expression, Var(expr) denotes the set of variables in an expression and I is an initialization function. It is defined from P into closed expressions such that:

$$\forall p \in P : [Type(I(p)) = C(p)MS]$$

III. Petri Machine: Simulating Turing Machines Using Petri Nets

As we said we take Colored Petri net with prioritized transitions and prove that it can be used to model Turing machines. We call the new model (i.e. a translated Petri Net that implements a Turing Machine) Petri Machine (PM). A Petri Machine is formally defined as a tuple $PM=(P,T,C,A,W,M_0,S_t)$ where P is an infinite set of places, T is an infinite set of transitions, C is a set of colors that can be assigned to the Petri Machine's tokens, A is an infinite set of arcs such that

$$A \subseteq (P \times T \times C) \cup (T \times P \times C)$$

W is a color function. It is defined from P into a color, M_0 is an initialization function, and S_t is an item of P that holds the current state of modeled Turing Machine (Turing State Place).

The remainder of this section focuses primarily on a detailed description of the simulating approach to present how different components of Turing machine will be simulated with the colored Petri net and how to use the simulating to build an efficient Petri machine.

A. States,tape Alphabet and Blank

States, tape alphabet and the blank character are distinguished in Turing Machine, so they must be the same in the Petri Machine as well. Therefore, all the tokens associated to these three components are given separate colors in the Petri Machine. Note that token with color number 1, however, is reserved for specific operations in the Petri Machine that will be discussed later. The below relation does the transformation of the above components to the color set, C, of Petri Machine:

$$C = f_c(Q) \cup f_c(\Gamma\Gamma \cup f_c(b) \cup f_c(1)$$

Where f_C is a function which is responsible for mapping from a text to a color.

B. The Tape and the Head

For any of locations of the head in TM, there is a separate corresponding place in the PM. Moreover, there is a token in the PM which corresponds to the head of TM. Meaning, whenever head moves, the token will be eventually transferred from one place to another. Thus, the set of locations of the tape and the head is a subset of all places of the Petri Machine:

$$f_p(Tape) \cup f_p(Head) \subseteq P$$

Where Tape is the set of cells of the tape and Head is the set of positions the Turing Machine's head will visit during the scan. The function f_P maps an element of the Tape and the Head to an element of places of Petri Machine. Note that PM always contains only one token which will get the color number 1 if it is placed in position(s) corresponding to the head of the Turing Machine.

C. Move Left and Right on the Tape

There are two places in the Petri Machine, called Shift-Left Control (SLC) and Shift-Right Control (SRC), whose job is to control left and right movements of the token between the Head places in PM, respectively. At each time, only one of the two places has the token with color 1. In order to move the token between the Head's places (i.e. the set defined as Head) to left or right, one must send the token to either one of the two SLC or SRC places, respectively.

In order to move the token from Head[i] to Head[i+1] (Shift Right) or Head[i+1] to Head[i] (Shift Left), there are two

DOI: 01.IJRTET.9.1.34

transitions, *Shift Left*[i] (SL[i]) and *Shift Right*[i] (SR[i]). The sources of *Shift Right*[i]'s input arcs are coming from *Shift Right Control* and Head[i] and destination of its output arc is Head[i+1]. Similarly, *Shift Left*[i]'s input arcs are coming from *Shift Left Control* and Head[i+1] positions while its output arc will go to the Head[i]. Note that, during the above left/ right shifts, all the arcs involved in the shifting operation will be passing the token with color 1. In Fig. 1 shift right and shift left for four Head Places is showný.

D. Read/Write from/to the Tape

To read/write from/to the tape, two places are needed to store the read/written token. We call these two places *Tape Data* and *New Tape Data*. In the Petri Machine, to every Tape[i], we add W new write transitions, named as Write[i][j], where W is the number of alphabets and 0 d" j < W. The added write transitions have input arcs coming from *New Tape Data* and Head[i] places and output arcs going to Tape[i] and Head[i]. For the New Tape Data then, the input arc carries the token with a color the same as the color of j^{th} element of alphabet, while the output arc also carries the token with the same color as the input arc. Using this technique, we are able to put in the Tape place the token with a color the same as the token in *New Tape Data*'s place. Therefore, for every place Head scans, we need to have as many transitions as the size of tape alphabet. Note that sending the token with color 1 to Head[i] place makes the token associated to Head to stay at its previous place.

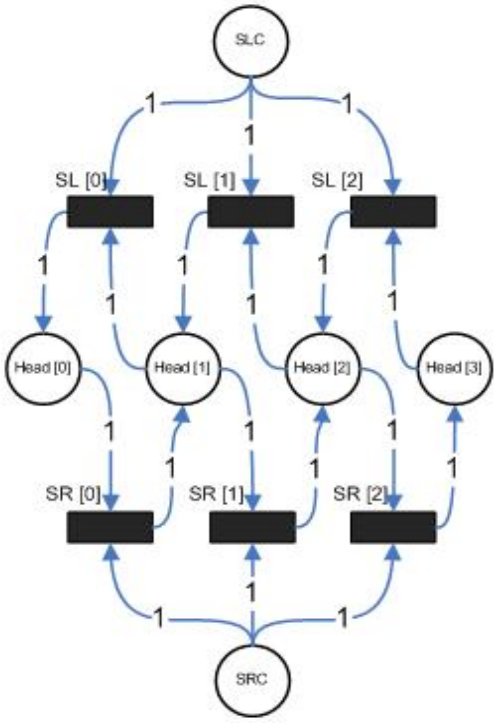

Figure 1. Shift right (SR) and shift left (SL) transitions for four head places

Likewise, in the Petri Machine, to every Tape[i], we add R new read transitions, named as Read[i][j], where R is the number of alphabets and 0 d" j < R. The added read transitions have input arcs coming from Tape[i] and Head[i] places and output arcs going to the *Tape Data* and Head[i]. Similar to Write[i][j], for the *Tape Data* then, the input arc carries the token with a color the same as the color of j^{th} element of alphabet, while the output arc also carries the token with the same color as the input arc, and sending the token with color 1 to Head[i] place makes the token associated to Head to stay at its previous place. In Fig. 2 two Write and two Read transitions are modeled.

Due to presentation and space limit only, we add the artificial transition * (asterisk) in order to demonstrate read and write transitions for all the places associated to the tape. Fig. 3 shows how * is used instead of the color set {a, b, c}.

E. Transition Function δ

In order to keep the state of Turing Machine at any time, there is a specific place in the Petri Machine, called Turing State, which keeps the token with a color related to the Turing Machine's state. For every transition in the Turing Machine's transition function, δ defined as $(q[i], \alpha) \rightarrow (q[j], \beta, R/L)$, we add a Check transition to the Petri Machine. Input arcs of the added Check transition(s) are coming from Tape Data and Turing State places while the output arcs are going to New Tape Data, Turing State and SRC (if head is moving to right)

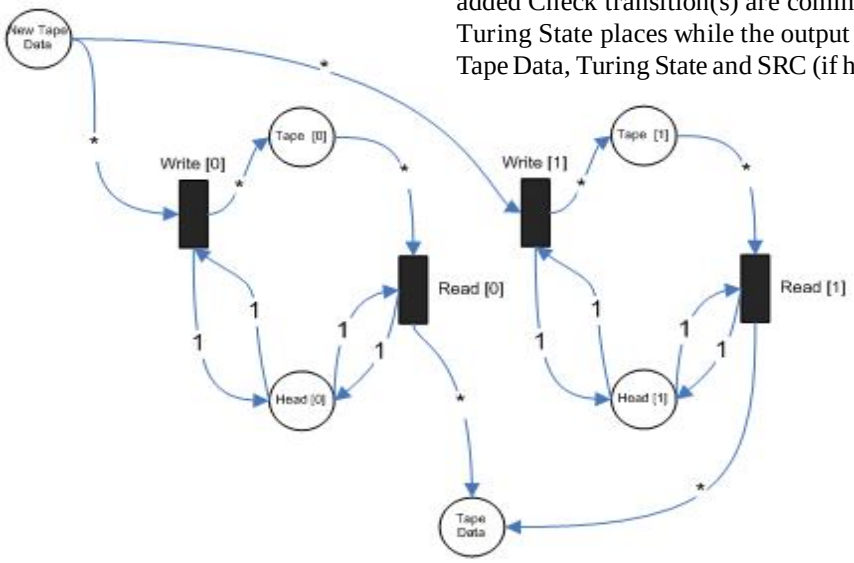

Figure 2. Modeling reading from/writing to the tape of a Turing Machine using Petri Machine

DOI: 01.IJRTET.9.1.34

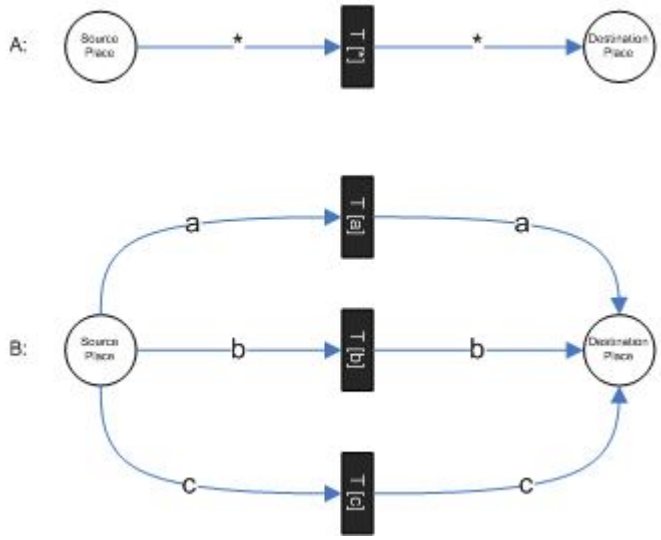

Figure 3. A: Star Transition and B: the original Petri Net for case A with color set {a, b, c}

or SLC (it head is moving to left) places. The Tape Data's input arc carries token with color, the Turing State's input arc carries one with color q[i], Turing State's output arc carries one with color q[j], and New Tape Data's output arc carries token with color. In Fig. 4 two Transitions are modeled. There two transitions are $\delta m : (q[i], a) \rightarrow (q[j], c, R)$ and $\delta n : (q[k], b) \rightarrow (q[l], d, L)$.

states (i.e. |Q|-|F|) and 0 d" i < F, 0 d" j < K. The *CISIF* transitions are used to consume the token(s) associated to the Turing State place. When no other transitions with priority greater than 2 were ready to fire, these added transitions would get fired and clear (empty) the Turing State place if it have the token with colors associated to the acceptance states, the set F, and finally, when no other transitions were ready to fire, *CISINF* transitions would get fired and clear (empty) the

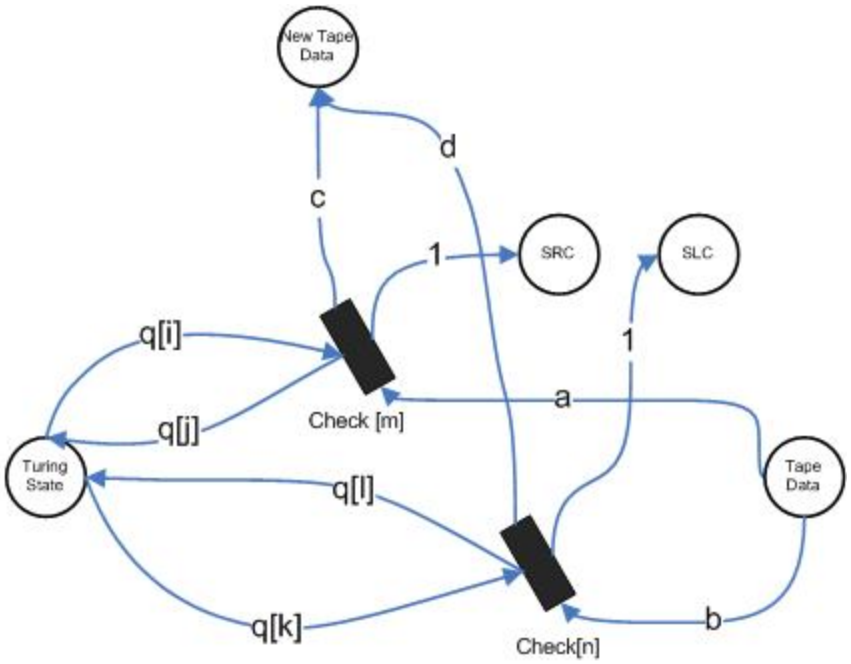

Figure 4. Modeling (q[i], α) $\rightarrow$ (q[j], β ,R/L) transition of Turing Machine using Petri Machine

F. Input Acceptance

To simulate the acceptance state of a Turing Machine in the Petri Machine, we add F number of CISIF[i] (*Check if State is Final* [i]) transitions and K number of CISINF[j] (*Check if State is not Final* [j]) transitions in the Petri Machine, where F is the number of Turing Machine's acceptance states (i.e. |F|), K is the number of Turing Machine's non acceptance Turing State place if it have the token with colors associated to the non acceptance states, the set Q-F. Note that the *CISIF[i]* transition have input arc comes from the Turing State and they carry token with a color associated to the i^{th} member of the set F and an output arc to *Accepted* place with color associated to 1, and the *CISINF[j]* transition have input arc comes from the Turing State and they carry token with a

DOI: 01.IJRTET.9.1.34

color associated to the j^{th} member of the set Q-F and an output arc to *Not Accepted* place with color associated to 1. ýIn Fig. 5 model of Input Acceptance is shown.

G. Prioritizing Transitions in Petri Machine

Priorities are assigned to every transition in a Petri Machine such that it enables modeling of different steps of operation of the target Turing Machine. In a Turing Machine, steps of operation are defined as Read, Write,

Check, Shift Left (SL) and Shift Right (SR). Following is a list of conditions must be met in order to lead to an appropriate prioritization of transitions in a Petri Machine.

1.There is only one token associated to every Head place of the Petri Machine (e.g. Head[i]), therefore Read transitions will have same priorities. For the same reason, Write transitions in the Petri Machine will have equal priorities as well.

2. Since, in every point of time, only one of the Shift Left Control or Shift Right Control places can keep the token, therefore all the SL or SR transitions will have the same priorities.

3.The token sitting in the Head[i] cannot be moved to its left (Head[i-1]) or right (Head[i+1]) places until the token in the New Tape Data place is moved out.

4.SLC and SRC places are to keep the state temporarily, so "reading" from the tape, while there is a token sitting in these two places, is forbidden.

5.The Check transition is to be executed whenever there is a token in Tape Data place but the New Tape Data, the SLC and SRC places are empty of token. Thus, before executing the Check transition, a Write and Shift (both left and right) transitions must be executed. In addition, Check transition must be followed by a Read transition as well, in order for the new token to be moved to the Tape Data place.

6.The *CISIF* and *CISINF* transitions should not get fired until all the Write, Read, Check, SL and SR transitions are able to be fired.

7.All of *Check* transitions have same priorities All of *CISIF* and *CISINF* transitions have same priorities.

Using the above conditions, Table I demonstrates the relative priorities for each transition in a Petri Machine against each other.

Note that relational operators are used in the table to show the priority of a transition in a row related to any other transitions in the columns. For example, a Read transition has lower priority than any of Write, Shift (right and left), and also Check transitions, while equalized with other Read transition and has even higher priority than *CISIF* and *CISINF* transitions. As stated before, columns 6 and 7 of Table I shows that all the main Petri Machine transitions have higher priority than *CISIF* and *CISINF* transitions, meeting condition 6 and 8 in the above list.

Summarizing the prioritization of transitions in Petri Machine, using Table I, we put the transitions in an order, from the lowest (*CISIF or CISINF*) to the highest priority (Write), as following: (*CISIF* and *CISINF)*, Read, Check, Shift (L/R), and finally Write.

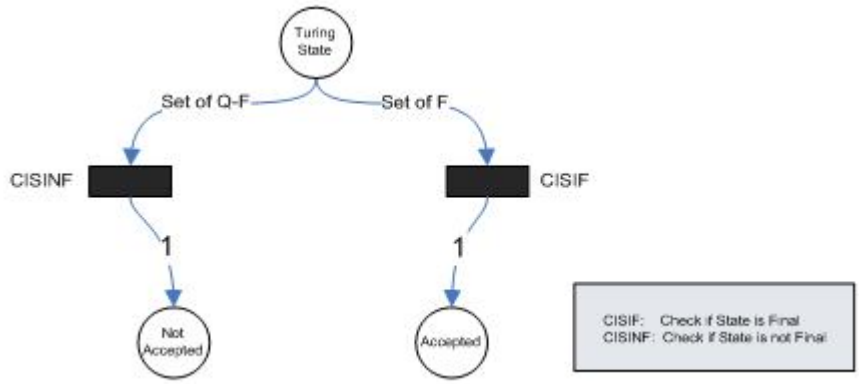

Figure 5. Modeling the input acceptance of Turing Machine using Petri Machine.

H. Putting It All Together

In this section, a sample Petri Machine is presented. By combining all the items mentioned in the two previous sections, it is possible to simulate a Turing Machine with a Petri Net which we called the result as Petri Machine. Fig. 6 depicts a general view of the Petri Machine. Note that symbol that is inside of each transition determines it's priority.

The following is how the Petri Machine (PM) works, according the Fig. 6 at the initial state, colored tokens corresponding to the Turing Machine's tape alphabet are put in every places of the PM. Head[0] contains a token with color 1, a token is put in place Turing State with color related to start state of Turing Machine (q_0). Moreover, tokens with color related to the input string's alphabets are put in order on the Petri Machine's tape places.

The first transition that ever fires is the Read[0] because there are initially only Read[0], *CISIF* and *CISINF* ready to be fired, which the first is chosen due to priority. With this first transition, the token related to Tape[0] is moved to *Tape Data*. Now, since Tape[0] has become empty, the transition Read[0] is no longer able to be fired. Then due to priority, the *Check* transition will be fired between the three transitions available: *Check*, *CISIF* and *CISINF*. According to the input/ output of the place where this *Check* transition happened,

DOI: 01.IJRTET.9.1.34

the token with new color will be moved to *New Tape Data* place and token with color 1 will be put in the *Shift Right/Left Control* place (based on the Turing Machine's transition).

At this point, all the *Shift, Write, CISIF* and *CISINF* will be ready to be fired. *Write* will fire due to priority which will move the token with new color to the current place. This causes the four transitions *Shift, Read, CISIF* and *CISINF* to be ready where only *Shift* will be fired due to priority. This *Shift* transition causes the token to be moved from Head[0] to Head[1]. This movement implies that the Turing Machine's head has moved one step.

Every repetition of the above procedure (cycle) technically simulates one Turing Machine's transition. The PM halts (exits the cycle) when none of the *Check* transitions is able to be fired. At the exit state of PM, the *CISIF* or *CISINF* transition (due to color of token that exist in *Turing State* place) will fire and clear the *Turing State* place.

When Petri Machine halts, it implies that the corresponding Turing Machine is also halted. However, when PM halts and *Accepted* place has a token with color 1, it means that the Turing Machine, and hence the Petri Machine as well, has accepted the input string, in the other hand if *Not Accepted* place has a token with color 1, it implies that the corresponding Turing Machine has halted at the position which is not a *final* state. ýIn Fig. 6ý, ýyou can see the final Petri machine.

TABLE I. RELATIVE PRIORITIES OF THE PETRI MACHINE TRANSITIONS AGAINST EACH OTHER

	Write	Shift R/L	Read	Check	CISIF	CISINF
Write	=	>	>	>	>	>
Shift R/L	<	=	>	>	>	>
Read	<	<	=	<	>	>
Check	<	<	>	=	>	>
CISIF	<	<	<	<	=	=
CISINF	<	<	<	<	=	=

IV. ANALYSIS OF TIME COMPLEXITY

As we said in section 2.1, a Turing machine M is defined precisely by $M = (Q, \Gamma, b, \delta, q_0, F)$. We suppose that *m* is the size of input string in the Turing machine, *n* is the size of Γ, *q* is the size of Q, *f* is the size of F and the number of existing rules in δ is *r*. The Turing Machine instruction for the execution requires a readý check, ýshift, clear and wait, So the execution of the Petri machine instruction needs five operations. The required time for the execution of these five operations depends on the time that we spend for finding transition with the most high priority. Based on the explanation, the number of required steps for the execution of the instruction in Petri machine calculates as the following.

Transition with the fifth priority (write instruction) requires the at most *m* time units .

Transition with the fourth priority (shift right/left instruction) for checking the transitions with the fifth priority requires the at most *m* time units and for finding transition related to itself requires at most $2(m-1)$ time units , so altogether require at most $(m + 2(m-1))$ time units .

Transition with the third priority (check instruction) require at most $(m+2(m-1) + r)$ time units .

Transition with the second priority require at most $(m+2(m-1) + r+n)$ time units .

Transition with the first priority will run only once for the entire input and requires at most $(m + 2(m-1) + r + n + f)$ time units .

We can conclude that if the Turing machine checks the input in K steps, the Petri machine will check it in $(K * (13m + 3r + 2n - 8) + f)$ steps. So if the Turing machine parameters such as the size of Γ are Polynomials, the required step in the resulting Petri machine will be polynomial coefficient of the required step in the Turing machine. It can be interpreted that we can use the Petri machine instead of the Turing machine for investigating whether the input string belongs to the language that we simulated it with a Turing machine and now with the correspondence Petri machine.

CONCLUSIONS

In this paper it is shown that modeling a Turing Machine using Colored Petri Net with prioritized transitions is efficient and possible. The presented model is called Petri Machine. We illustrated that transitions with the same priority in Petri Net can be used for modeling Non Deterministic Turing Machine (NDTM), therefore all tools and formulas used for Colored Petri Net are applicable to modeling Turing Machine as well.

Moreover, due to the existence of priority amongst transitions, one can effectively take advantage of having (arbitrary) transitions with equalized priorities in order to use Petri Nets for modeling (i.e. simulating) different randomized algorithms. Note that in order to use Petri Nets to model problems, which are otherwise of most interest to be solved by Turing Machines, it is not required to use a specific form of Petri Machine. Therefore, having such a degree of freedom in choice will help us use Petri Nets more efficiently to model more diversified range of computation problems.

REFERENCES

[1] T. Agerwala and M. Flynn, "Comments on capabilities, limitations and "correctness" of Petri nets," Proceedings of the 1st annual symposium on Computer architecture, p.81-86,December 09-11, 1973

[2] T. Agerwala, " A complete model for representtng the coordination of asynchronous processes," Hopkins Computer Research Report No. 32, Computer Science Program, Johns Hopkins Univ., Baltimore, Md., July 1974, 58 pp.

DOI: 01.IJRTET.9.1.34

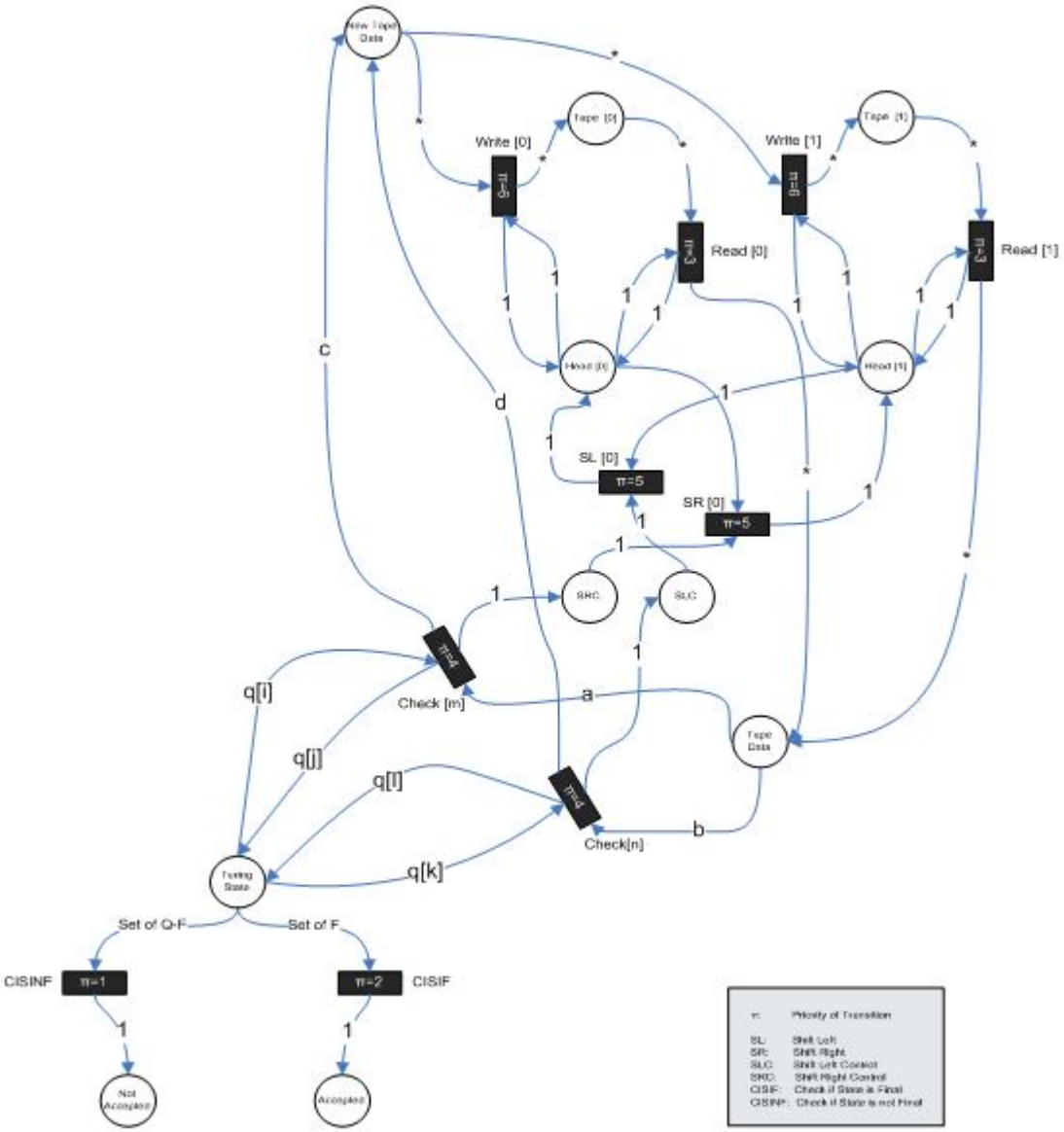

Figure 6. Overall view of Petri Machine

[3] A. Turing, "On Computable Numbers with an Application to the Entscheidungsproblem," Proceedings of the London Math Society, 2 (42), pp. 173-198, 1936.

[4] A. Church, "A set of Postulates for the Foundation of Logic," Annals of Mathematics, second series, vol 33, pp. 346-366, 1932.

[5] S.C. Kleene, "A Theory of Positive Integers in Formal Logic," American Journal of Mathematics, vol 57, 153-173, 219-244, 1935.

[6] J. L. Peterson, Petri Net Theory and the Modeling of Systems," Prentice Hall, 1981.

[7] T. Murata, "Petri Nets: Properties, Analysis and Applications," Proceedings of the IEEE, Vol.77, pp.541-580, 1989.

[8] K. Jensen, "Coloured Petri Nets, Basic concepts, Analysis Methods and Practical Use," Springer 1997.

DOI: 01.IJRTET.9.1.34